Modelling and Analysing Concurrent Systems

RIO 2023 Summer School of Informatics
Rio Cuarto, Argentina; February 13–17, 2023

Lecture 4: Application to Mutual-Exclusion Protocols

Thomas Noll
Software Modelling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-22-23/rio/

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Outline of Lecture 4

Peterson's Mutual Exclusion Algorithm

- **Goal:** ensuring exclusive access to non-shared resources
- Here: two competing processes P_1, P_2 and shared variables
  - b_1, b_2 (Boolean, both initially false) – b_i indicates that P_i wants to enter critical section
  - k (in $\{1, 2\}$, arbitrary initial value) – index of prioritised process
- P_i uses local variable $j := 2 - i$ (index of other process)

Peterson's Mutual Exclusion Algorithm

- **Goal:** ensuring exclusive access to non-shared resources
- Here: two competing processes P_1, P_2 and shared variables
  - b_1, b_2 (Boolean, both initially false) – b_i indicates that P_i wants to enter critical section
  - k (in $\{1, 2\}$, arbitrary initial value) – index of prioritised process
- P_i uses local variable $j := 2 - i$ (index of other process)

Algorithm 4.1 (Peterson's algorithm for P_i)

while true do
    "non-critical section";
    $b_i :=$ true;
    $k := j$;
    while $b_j \wedge k = j$ do skip end;
    "critical section";
    $b_i :=$ false;
end

Representing Shared Variables in CCS

- Not directly expressible in CCS (communication by handshaking)
- Idea: consider variables as processes that communicate with environment by processing read/write requests

Representing Shared Variables in CCS

- Not directly expressible in CCS (communication by handshaking)
- Idea: consider variables as processes that communicate with environment by processing read/write requests

Example 4.2 (Shared variables in Peterson's algorithm)

- Encoding of b_1 with two (process) states B_{1t} (value tt) and B_{1f} (value ff)
- Read access along ports $b1rt$ (in state B_{1t}) and $b1rf$ (in state B_{1f})
- Write access along ports $b1wt$ and $b1wf$ (in both states)

Representing Shared Variables in CCS

- Not directly expressible in CCS (communication by handshaking)
- Idea: consider variables as processes that communicate with environment by processing read/write requests

Example 4.2 (Shared variables in Peterson's algorithm)

- Encoding of b_1 with two (process) states B_{1t} (value tt) and B_{1f} (value ff)
- Read access along ports $b1rt$ (in state B_{1t}) and $b1rf$ (in state B_{1f})
- Write access along ports $b1wt$ and $b1wf$ (in both states)
- Possible behaviours:

$$B_{1f} = \overline{b1rf}.B_{1f} + b1wf.B_{1f} + b1wt.B_{1t}$$
$$B_{1t} = \overline{b1rt}.B_{1t} + b1wf.B_{1f} + b1wt.B_{1t}$$

Representing Shared Variables in CCS

- Not directly expressible in CCS (communication by handshaking)
- Idea: consider variables as processes that communicate with environment by processing read/write requests

Example 4.2 (Shared variables in Peterson's algorithm)

- Encoding of b_1 with two (process) states B_{1t} (value tt) and B_{1f} (value ff)
- Read access along ports $b1rt$ (in state B_{1t}) and $b1rf$ (in state B_{1f})
- Write access along ports $b1wt$ and $b1wf$ (in both states)
- Possible behaviours:

$$B_{1f} = \overline{b1rf}.B_{1f} + b1wf.B_{1f} + b1wt.B_{1t}$$
$$B_{1t} = \overline{b1rt}.B_{1t} + b1wf.B_{1f} + b1wt.B_{1t}$$

- Similarly for b_2 and k:

$$B_{2f} = \overline{b2rf}.B_{2f} + b2wf.B_{2f} + b2wt.B_{2t}$$
$$B_{2t} = \overline{b2rt}.B_{2t} + b2wf.B_{2f} + b2wt.B_{2t}$$

$$K_1 = \overline{kr1}.K_1 + kw1.K_1 + kw2.K_2$$
$$K_2 = \overline{kr2}.K_2 + kw1.K_1 + kw2.K_2$$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

while true do
  "non-critical section";
  $b_i :=$ true;
  $k := j$;
  while $b_j \land k = j$ do skip end;
  "critical section";
  $b_i :=$ false;
end

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

while true do
  "non-critical section";
  $b_i :=$ true;
  $k := j$;
  while $b_j \wedge k = j$ do skip end;
  "critical section";
  $b_i :=$ false;
end

CCS representation

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11}\\
P_{11} &= b2rf.P_{12} + \\
&\quad\; b2rt.(kr1.P_{12} + kr2.P_{11})\\
P_{12} &= enter1.exit1.\overline{b1wf}.P_1
\end{aligned}
$$

$$
\begin{aligned}
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21}\\
P_{21} &= b1rf.P_{22} + \\
&\quad\; b1rt.(kr1.P_{21} + kr2.P_{22})\\
P_{22} &= enter2.exit2.\overline{b2wf}.P_2
\end{aligned}
$$

$$Peterson = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$$

for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

while true do
  "non-critical section";
  b_i := true;
  $k := j$;
  while $b_j \wedge k = j$ do skip end;
  "critical section";
  b_i := false;
end

CCS representation

$$\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11} \\
P_{11} &= b2rf.P_{12} + \\
&\quad\; b2rt.(kr1.P_{12} + kr2.P_{11}) \\
P_{12} &= enter1.exit1.\overline{b1wf}.P_1
\end{aligned}$$

$$\begin{aligned}
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21} \\
P_{21} &= b1rf.P_{22} + \\
&\quad\; b1rt.(kr1.P_{21} + kr2.P_{22}) \\
P_{22} &= enter2.exit2.\overline{b2wf}.P_2
\end{aligned}$$

$$Peterson = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$$

for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

while true do
  "non-critical section";
  $b_i :=$ true;
  $k := j$;
  while $b_j \wedge k = j$ do skip end;
  "critical section";
  $b_i :=$ false;
end

CCS representation

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11} \\
P_{11} &= b2rf.P_{12} + \\
&\quad\ b2rt.(kr1.P_{12} + kr2.P_{11}) \\
P_{12} &= enter1.exit1.\overline{b1wf}.P_1
\end{aligned}
$$

$$
\begin{aligned}
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21} \\
P_{21} &= b1rf.P_{22} + \\
&\quad\ b1rt.(kr1.P_{21} + kr2.P_{22}) \\
P_{22} &= enter2.exit2.\overline{b2wf}.P_2
\end{aligned}
$$

$$Peterson = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$$

for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

while true do
  "non-critical section";
  $b_i :=$ true;
  $k := j$;
  while $b_j \wedge k = j$ do skip end;
  "critical section";
  $b_i :=$ false;
end

CCS representation

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11} \\
P_{11} &= b2rf.P_{12} + \\
&\quad\; b2rt.(kr1.P_{12} + kr2.P_{11}) \\
P_{12} &= enter1.exit1.\overline{b1wf}.P_1
\end{aligned}
$$

$$
\begin{aligned}
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21} \\
P_{21} &= b1rf.P_{22} + \\
&\quad\; b1rt.(kr1.P_{21} + kr2.P_{22}) \\
P_{22} &= enter2.exit2.\overline{b2wf}.P_2
\end{aligned}
$$

$$Peterson = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$$

$$\text{for } L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

while true do
  "non-critical section";
  $b_i :=$ true;
  $k := j$;
  while $b_j \land k = j$ do skip end;
  "critical section";
  $b_i :=$ false;
end

CCS representation

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11} \\
P_{11} &= b2rf.P_{12} + \\
&\quad\; b2rt.(kr1.P_{12} + kr2.P_{11}) \\
P_{12} &= enter1.exit1.\overline{b1wf}.P_1
\end{aligned}
$$

$$
\begin{aligned}
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21} \\
P_{21} &= b1rf.P_{22} + \\
&\quad\; b1rt.(kr1.P_{21} + kr2.P_{22}) \\
P_{22} &= enter2.exit2.\overline{b2wf}.P_2
\end{aligned}
$$

$$Peterson = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$$

$$\text{for } L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

while true do
    "non-critical section";
    $b_i :=$ true;
    $k := j$;
    while $b_j \wedge k = j$ do skip end;
    "critical section";
    $b_i :=$ false;
end

CCS representation

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11} \\
P_{11} &= b2rf.P_{12} + \\
&\quad\; b2rt.(kr1.P_{12} + kr2.P_{11}) \\
P_{12} &= enter1.exit1.\overline{b1wf}.P_1
\end{aligned}
$$

$$
\begin{aligned}
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21} \\
P_{21} &= b1rf.P_{22} + \\
&\quad\; b1rt.(kr1.P_{21} + kr2.P_{22}) \\
P_{22} &= enter2.exit2.\overline{b2wf}.P_2
\end{aligned}
$$

$$Peterson = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$$

for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

while true do
  "non-critical section";
  $b_i :=$ true;
  $k := j$;
  while $b_j \land k = j$ do skip end;
  "critical section";
  $b_i :=$ false;
end

CCS representation

$$\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11}\\
P_{11} &= b2rf.P_{12} + {}\\
&\quad\; b2rt.(kr1.P_{12} + kr2.P_{11})\\
P_{12} &= enter1.exit1.\overline{b1wf}.P_1
\end{aligned}$$

$$\begin{aligned}
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21}\\
P_{21} &= b1rf.P_{22} + {}\\
&\quad\; b1rt.(kr1.P_{21} + kr2.P_{22})\\
P_{22} &= enter2.exit2.\overline{b2wf}.P_2
\end{aligned}$$

$$Peterson = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$$

$$\text{for } L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$$

Outline of Lecture 4

Modelling Mutual Exclusion Algorithms

Evaluating the CCS Model

Verifying Properties by Model Checking

Verifying Mutual Exclusion by Bisimulation Checking

The End

Obtaining the LTS using CAAL I

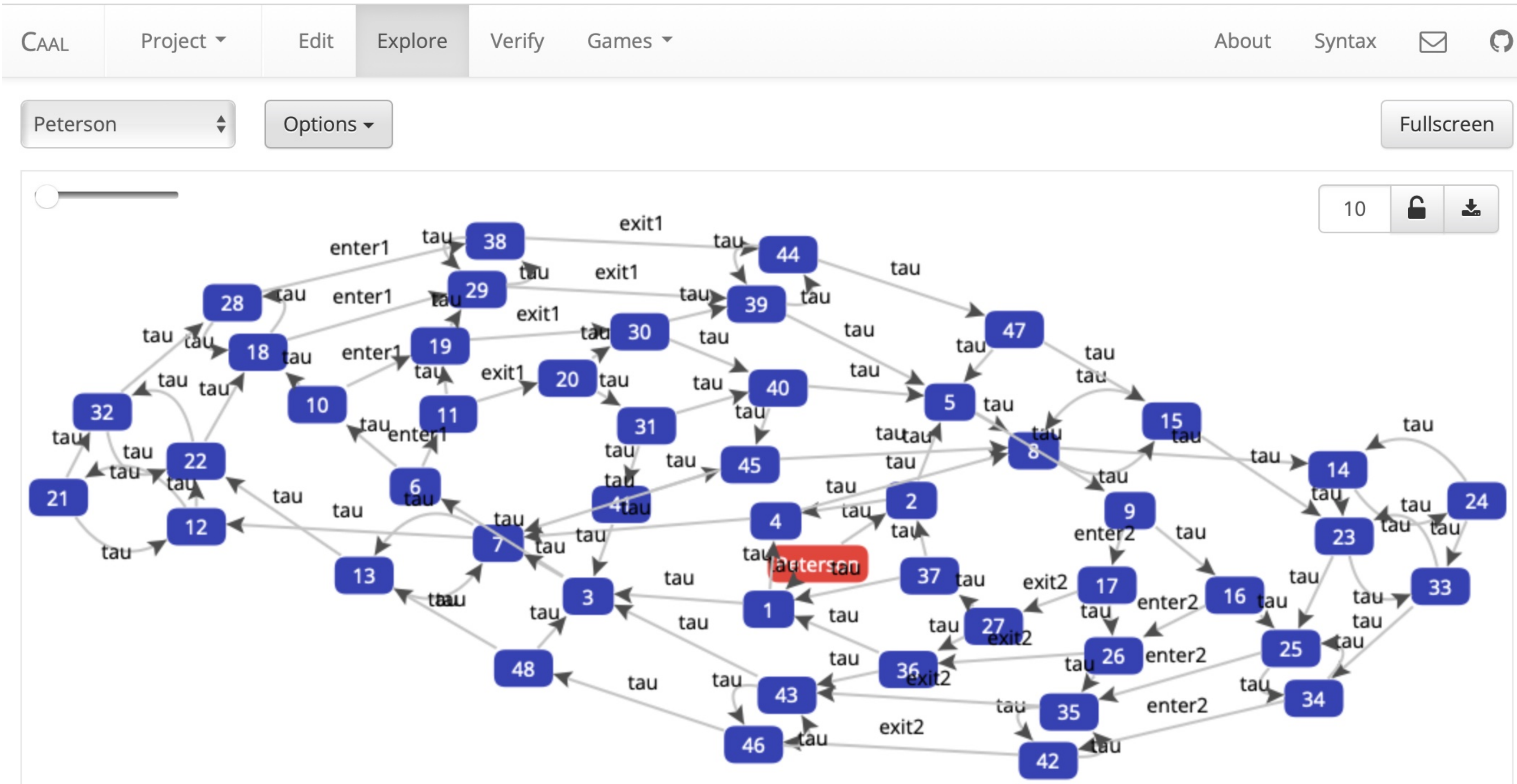
CAAL | Project | Edit | Explore | Verify | Games | About | Syntax

MutEx

Parse | CCS | TCCS | 16

```
* Peterson's algorithm for mutual exclusion.
* See Chapter 7 of "Reactive Systems" for a full description.

B1f = 'b1rf.B1f + b1wf.B1f + b1wt.B1t;
B1t = 'b1rt.B1t + b1wf.B1f + b1wt.B1t;

B2f = 'b2rf.B2f + b2wf.B2f + b2wt.B2t;
B2t = 'b2rt.B2t + b2wf.B2f + b2wt.B2t;

K1 = 'kr1.K1 + kw1.K1 + kw2.K2;
K2 = 'kr2.K2 + kw1.K1 + kw2.K2;

P1 = 'b1wt.'kw2.P11;
P11 = b2rf.P12 + b2rt.(kr2.P11 + kr1.P12);
P12 = enter1.exit1.'b1wf.P1;

P2 = 'b2wt.'kw1.P21;
P21 = b1rf.P22 + b1rt.(kr1.P21 + kr2.P22);
P22 = enter2.exit2.'b2wf.P2;

set L = {b1rf, b2rf, b1rt, b2rt, b1wf, b2wf, b1wt, b2wt, kr1, kr2, kw1, kw2};
Peterson = (P1 | P2 | B1f | B2f | K1) \ L;

MutExCCS = enter1.exit1.MutExCCS + enter2.exit2.MutExCCS;
```

Obtaining the LTS using CAAL II



Source	Action	Target
Peterson	tau	1
Peterson	tau	2

Outline of Lecture 4

The Mutual Exclusion Property

- **Done:** formal description of Peterson's algorithm
- **To do:** analysing its behaviour
- **Question:** what does "ensuring mutual exclusion" formally mean?

The Mutual Exclusion Property

- **Done:** formal description of Peterson's algorithm
- **To do:** analysing its behaviour
- **Question:** what does "ensuring mutual exclusion" formally mean?

Mutual exclusion

At no point in the execution of the algorithm, processes P_1 and P_2 will both be in their critical section at the same time.

Equivalently:
It is always the case that either P_1 or P_2 or both are not in their critical section.

Specifying Mutual Exclusion in HML

Mutual exclusion

It is always the case that either P_1 or P_2 or both are not in their critical section.

Specifying Mutual Exclusion in HML

Mutual exclusion

It is always the case that either P_1 or P_2 or both are not in their critical section.

Observations:

- Mutual exclusion is an invariance property ("always")
- P_i is in its critical section iff action *exit i* is enabled

Specifying Mutual Exclusion in HML

Mutual exclusion

It is always the case that either P_1 or P_2 or both are not in their critical section.

Observations:

- Mutual exclusion is an invariance property ("always")
- P_i is in its critical section iff action *exit i* is enabled

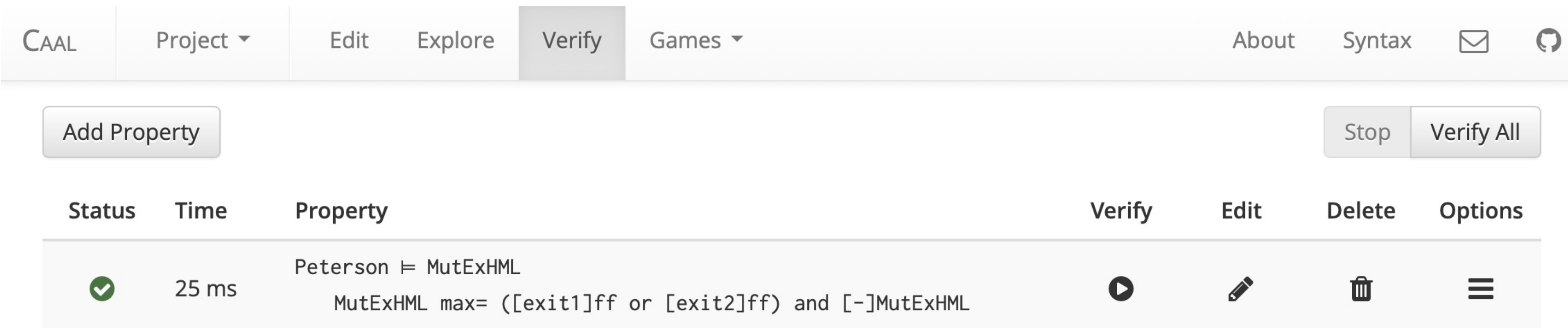
Mutual exclusion in HML

$$
\begin{aligned}
MutExHML &:= Inv(NotBoth) \\
Inv(F) &\stackrel{max}{=} F \wedge [Act]Inv(F) \qquad \text{(cf. Example 3.11)} \\
NotBoth &:= [exit1]\mathrm{ff} \vee [exit2]\mathrm{ff}
\end{aligned}
$$

Model Checking Mutual Exclusion

- Using CAAL Tool
- Supports property specifications by recursive HML formulae:

```
MutExHML max= ([exit1]ff or [exit2]ff) and [-]MutExHML;
```



Absence of Deadlocks

Absence of deadlocks

It is always the case that the system can progress, i.e., perform any action.

Absence of Deadlocks

Absence of deadlocks

It is always the case that the system can progress, i.e., perform any action.

Absence of deadlocks in HML

$$\begin{aligned} \mathit{NoDeadlocks} &:= \mathit{Inv}(\mathit{CanProgress}) \\ \mathit{Inv}(F) &\stackrel{\mathit{max}}{=} F \land [\mathit{Act}]\mathit{Inv}(F) \\ \mathit{CanProgress} &:= \langle \mathit{Act} \rangle \mathrm{tt} \end{aligned}$$

Fairness

Fairness

Whenever a process requires access to the critical section, it will eventually be granted.

Fairness

Fairness

Whenever a process requires access to the critical section, it will eventually be granted.

Observation: requires nesting of recursive formulae

- outer: “whenever” $\Rightarrow$ invariant $\Rightarrow$ greatest fixed point
- inner: “eventually” $\Rightarrow$ to be ensured after finitely many steps $\Rightarrow$ least fixed point

Fairness

Fairness

Whenever a process requires access to the critical section, it will eventually be granted.

Observation: requires nesting of recursive formulae

- outer: “whenever” $\Rightarrow$ invariant $\Rightarrow$ greatest fixed point
- inner: “eventually” $\Rightarrow$ to be ensured after finitely many steps $\Rightarrow$ least fixed point

Fairness in HML (only first process)

$$
\begin{aligned}
Fair_1 &:= Inv(EventuallyLeaves_1) \\
Inv(F) &\overset{max}{=} F \wedge [Act]Inv(F) \\
EventuallyLeaves_1 &:= Evt(Exits_1) \\
Evt(G) &\overset{min}{=} G \vee [Act]Evt(G) \\
Exits_1 &:= \langle exit1 \rangle \mathrm{tt}
\end{aligned}
$$

Outline of Lecture 4

Specifying Mutual Exclusion in CCS

- **Goal:** express desired behaviour of algorithm as an “abstract” CCS process

Specifying Mutual Exclusion in CCS

- **Goal:** express desired behaviour of algorithm as an "abstract" CCS process
- Intuitively:
  (1) Initially, either P_1 or P_2 can enter its critical section.
  (2) Once this happened, the other process cannot enter the critical section before the first has exited it .

Specifying Mutual Exclusion in CCS

- **Goal:** express desired behaviour of algorithm as an "abstract" CCS process
- Intuitively:
  (1) Initially, either P_1 or P_2 can enter its critical section.
  (2) Once this happened, the other process cannot enter the critical section before the first has exited it .

Mutual exclusion in CCS

$$MutExCCS = enter1.exit1.MutExCCS + enter2.exit2.MutExCCS$$

Specifying Mutual Exclusion in CCS

- **Goal:** express desired behaviour of algorithm as an "abstract" CCS process
- Intuitively:
  (1) Initially, either P_1 or P_2 can enter its critical section.
  (2) Once this happened, the other process cannot enter the critical section before the first has exited it .

Mutual exclusion in CCS

$$\mathit{MutExCCS} = \mathit{enter1}.\mathit{exit1}.\mathit{MutExCCS} + \mathit{enter2}.\mathit{exit2}.\mathit{MutExCCS}$$

- Weak bisimulation does *not* hold as *Peterson* satisfies additional fairness constraints (prioritisation via variable k).
- However, *Peterson* and *MutExSpec* share the same weak traces (i.e., action sequences ignoring τ-transitions).

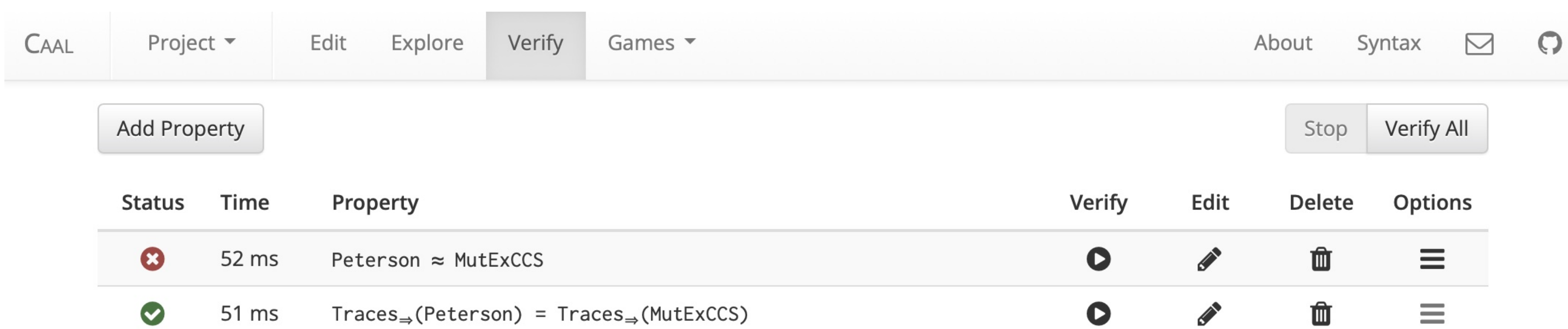
Specifying Mutual Exclusion in CCS

- **Goal:** express desired behaviour of algorithm as an “abstract” CCS process
- Intuitively:
  (1) Initially, either P_1 or P_2 can enter its critical section.
  (2) Once this happened, the other process cannot enter the critical section before the first has exited it .

Mutual exclusion in CCS

$$\mathit{MutExCCS} = \mathit{enter1}.\mathit{exit1}.\mathit{MutExCCS} + \mathit{enter2}.\mathit{exit2}.\mathit{MutExCCS}$$

- Weak bisimulation does *not* hold as *Peterson* satisfies additional fairness constraints (prioritisation via variable k).
- However, *Peterson* and *MutExSpec* share the same weak traces (i.e., action sequences ignoring τ-transitions).



Outline of Lecture 4

Modelling Mutual Exclusion Algorithms

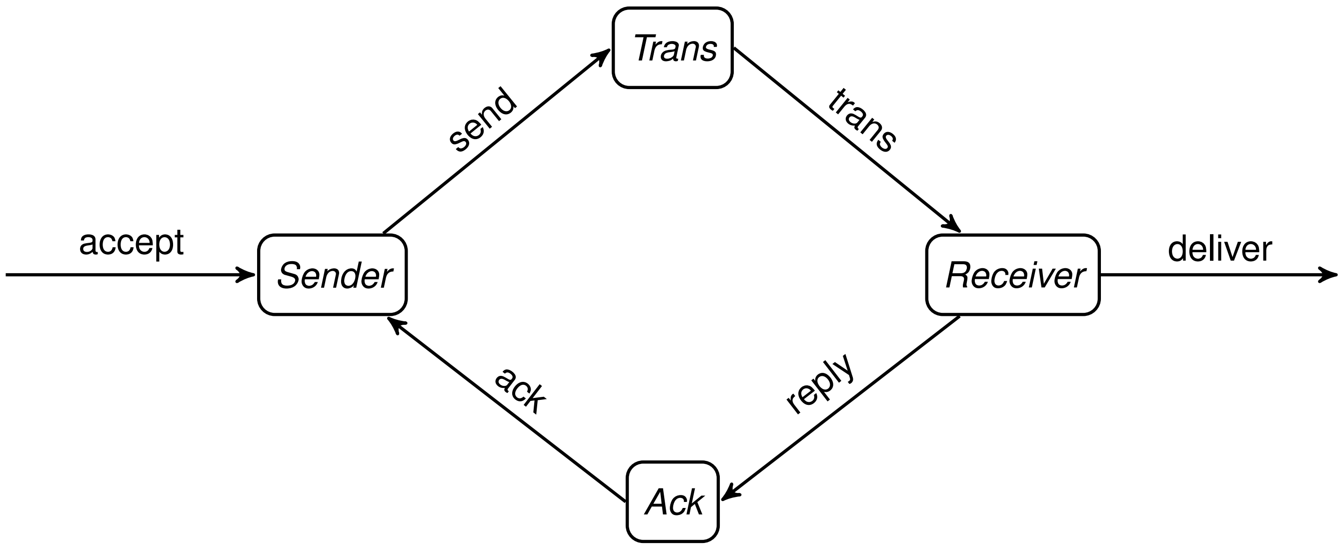
Evaluating the CCS Model

Verifying Properties by Model Checking

Verifying Mutual Exclusion by Bisimulation Checking

The End

Homework: The Alternating-Bit Protocol



To do (using CAAL):

- Implementation as CCS process definition
- Abstract specification in CCS and bisimilarity checking
- Model checking for deadlocks and livelocks
- Deliverable: short experience report with description of outcomes
- Deadline: March 31, 2023
- More details in `https://moves.rwth-aachen.de/wp-content/uploads/abp.pdf`